Combining Constraint Programming and Local Search for Job-Shop Scheduling

J. Christopher Beck, T. K. Feng
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{jcb,tkfeng}@mie.utoronto.ca

Jean-Paul Watson
Discrete Math and Complex Systems Department
Sandia National Laboratories, Albuquerque, New Mexico, USA
jwatson@sandia.gov

Since their introduction, local search algorithms have consistently represented the state-of-the-art in solution techniques for the classical job-shop scheduling problem. This is despite the availability of powerful search and inference techniques for scheduling problems developed by the constraint programming community. In this paper, we introduce a simple hybrid algorithm for job-shop scheduling that leverages both the fast, broad search capabilities of modern tabu search algorithms and the scheduling-specific inference capabilities of constraint programming. The hybrid algorithm significantly improves the performance of a state-of-the-art tabu search algorithm for the job-shop problem, and represents the first instance in which a constraint programming algorithm obtains performance competitive with the best local search algorithms. Further, the variability in solution quality obtained by the hybrid is significantly lower than that of pure local search algorithms. Beyond performance demonstration, we perform a series of experiments that provide insights into the roles of the two component algorithms in the overall performance of the hybrid.

Key words: Scheduling, Tabu Search, Constraint Programming, Hybrid Algorithms.
History: Submitted June TBD, 2009.

1. Introduction

Local search algorithms for the traditional makespan-minimization formulation of the job-shop scheduling problem (JSP) have dominated the state-of-the-art for at least the past 15 years. These include Nowicki and Smutnicki's landmark TSAB tabu search algorithm (Nowicki and Smutnicki, 1996), Balas and Vazacopoulos' guided local search algorithm (Balas and Vazacopoulos, 1998), Nowicki and Smutnicki's follow-on *i*-TSAB tabu search algorithm

(Nowicki and Smutnicki, 2005), and most recently Zhang et al.'s hybrid tabu search / simulated annealing algorithm (Zhang et al., 2008). These algorithms are all built upon a foundation of one or more powerful, problem-specific move operators, which are able to efficiently identify promising feasible and high-quality solutions in the neighborhood of a given solution. Metaheuristic search strategies then leverage these move operators to perform global search for minimal-cost solutions; the complexity of these strategies ranges from simple tabu search (in the case of TSAB) to highly intricate hybridizations of tabu search, path relinking, and elite pool maintenance schemes (in the case of i-TSAB).

Perhaps somewhat paradoxically, constraint programming (CP) algorithms are more commonly used than their local search counterparts to obtain solutions to real-world scheduling problems, e.g., using ILOG's Scheduler software library (Scheduler, 2007). This is widely attributed to a combination of the ability to easily incorporate various idiosyncratic "side" constraints that are pervasive in real-world scheduling problems (such constraints can require significant redesign of local search algorithms) and to effectively deduce, via powerful domain-specific constraint propagators, the implications of various scheduling decisions. However, despite the volume of research dedicated to the development of scheduling-specific constraint propagation and search techniques (e.g., see Baptiste et al. (2001); Beck and Fox (2000)), the performance of CP algorithms on the traditional JSP has significantly lagged that of their local search counterparts. To date, the strongest CP-based algorithm is solution-guided search (Beck, 2007), although the performance of even this algorithm lags that of modern tabu search algorithms for the JSP (Watson et al., 2006) in terms of both time and final solution quality.

Hybridization of local search and CP on JSPs without side constraints does not, therefore, immediately appear to be a promising research direction. However, the following two unexplored aspects of these algorithms motivate the line of research developed in this paper:

- The strong propagation techniques in CP are most efficient in constrained search states. That is, the polynomial time inference algorithms are more likely to be able to find implied constraints, and to consequently reduce the search space, in states that are already highly constrained. When a good solution has been found, strong "back-propagation" from the upper bound on the makespan results in such a highly constrained search state. Therefore, we conjecture that while CP is unable to competitively find good solutions on its own, once given a good solution, it may be able to improve on it more quickly than a local search approach.

- A popular conceptualization of the power of modern local search algorithms is that they balance intensification with diversification (Watson et al., 2006). Intensification, which can loosely be understood as searching "near" an existing good solution, is often implemented by repeatedly restarting search from a good solution that has been found previously. Diversification, in contrast, tries to distribute the search effort to unexplored areas of the search space. It is often implemented by maintaining a varied set of promising solutions and combining them in a variety of ways, such as via path relinking (Glover et al., 2003). However, modern tabu search algorithms seem to do a relatively poor job of intensification. Watson (2005) showed that after a relatively small number of iterations after restarting search from a good solution, tabu search is a considerable distance from the starting solution. Further, *a posteriori* analysis of algorithmic traces indicates that tabu search often fails to locate high-quality solutions that are quite close to previously identified solutions. In contrast, solution-guided constructive search performs a much more focused search around its guiding solution (Beck, 2007). Therefore, we conjecture that improved performance may result from using CP to strongly intensify search around a diverse set of high-quality solutions generated by tabu search.

The remainder of this paper is organized as follows. We begin in Section 2 with a brief discussion of the job-shop scheduling problem, the benchmark instances used in our analysis, the foundational algorithms for our hybrid approach, and a discussion of previous work on algorithm switching hybrids. Our simple hybrid is described in Section 3. Section 4 outlines our computational experiments, which are subsequently detailed in Sections 5 through 8. We compare the performance of our best parameterization of the hybrid algorithm with the state-of-the-art in Section 9. Section 10 details some implications of our results, followed by our conclusions in Section 11. The basic idea of our hybrid algorithm was previously explored by Watson and Beck (2008). This paper represents a significant extension in terms of experimental methodology, parameterization, and analysis; in particular, the notion of switching and the experiments reported in Sections 5 through 8 are all novel contributions.

2. Background, Problems, and Algorithms

In this section, we provide the context for our work: the problem and benchmark instances, the "pure" algorithms used as a basis for our hybrid approach, and details of previous work on related hybrid algorithms.

2.1 Problem Description and Benchmark Instances

We consider the well-known $n \times m$ static, deterministic JSP in which n jobs must be processed exactly once on each of m machines (Błażewicz et al., 1996). Each job i $(1 \leq i \leq n)$ is routed through each of the m machines in a pre-defined order π_i, where $\pi_i(j)$ denotes the jth machine $(1 \leq j \leq m)$ in the routing order of job i. The processing of job i on machine $\pi_i(j)$ is denoted o_{ij} and is called an operation. An operation o_{ij} must be processed on machine $\pi_i(j)$ for an integral duration $\tau_{ij} > 0$. Once an operation is initiated, processing cannot be pre-empted and concurrency on individual machines is not allowed, i.e., the machines are unit-capacity resources. For $2 \leq j \leq m$, o_{ij} cannot begin processing until $o_{i(j-1)}$ has completed processing. The scheduling objective is to minimize the makespan C_{max}, i.e., the maximal completion time of the last operation of any job. Makespan-minimization for the JSP is NP-hard for $m \geq 2$ and $n \geq 3$ (Garey et al., 1976).

An instance of the $n \times m$ JSP is uniquely defined by the set of nm operation durations τ_{ij} and n job routing orders π_i. In nearly all benchmark instances, the τ_{ij} are uniformly sampled from the interval $[1, 99]$, while the π_i are given by random permutations of the integer sequence $1, \ldots, m$. Our experimental results are generated using a subset of Taillard's well-known benchmark instances, specifically those labeled `ta11` through `ta50` (Taillard, 1993). This subset contains 10 instances of each of the following problem sizes: 20×15, 20×20, 30×15, and 30×20. We have selected these instances because they are widely studied, are known to be very challenging, and have "headroom" for improvement in best-known makespans. For these same reasons, we ignore the easier instances in Taillard's problem suite, in addition to many historical instances (e.g., the "ft", "la", and "orb" instances) for which modern JSP algorithms can consistently locate optimal solutions.

2.2 Iterated Simple Tabu Search

Beginning with an early approach by Taillard (1989), tabu search algorithms have consistently represented the state-of-the-art in obtaining high-quality solutions to the JSP. A variety of researchers have introduced tabu search algorithms of ever-increasing effectiveness and complexity. Specific algorithmic advances of note in this progression include the introduction of (1) the highly restrictive *N5* critical path-based move operator (Nowicki and Smutnicki, 1996), (2) search intensification mechanisms in conjunction with sets of "elite" or high-quality solutions (Nowicki and Smutnicki, 1996), and (3) search diversification mechanisms in the form of path relinking (Nowicki and Smutnicki, 2005). These components are simultaneously embodied in Nowicki and Smutnicki's *i*-TSAB algorithm, which has represented the state-of-the-art since 2003. With the exception noted below, the sole competitor is a hybrid tabu search / simulated annealing algorithm introduced by Zhang et al. (2008). The Zhang et al. algorithm uses simulated annealing to generate an initial set of elite solutions, which are then processed via tabu search-driven intensification. The primary differences between the Zhang et al. algorithm and *i*-TSAB are the lack of an explicit diversification mechanism (path relinking is used in *i*-TSAB) and the use of the *N6* move operator introduced by Balas and Vazacopoulos (1998) in the case of Zhang et al.

Although remarkably effective, *i*-TSAB is an extremely intricate and complex algorithm. Such complexity is a significant drawback to researchers, as in practice it impedes reproducibility, adoption, and subsequent study. In the specific case of *i*-TSAB, its intricacy makes it difficult to assess the contribution of the various algorithmic components to its overall performance. Toward this goal, we previously introduced a simplified version of *i*-TSAB called iterated simple tabu search (*i*-STS) (Watson et al., 2006), which contains the key algorithmic ingredients of *i*-TSAB while reducing the overall complexity and maintaining near-equivalent performance. **Pseudo-code for *i*-STS is provided in Figure 1. A summary description of the algorithm follows; full details are provided in Watson et al. (2006)**.

A basic tabu search lies at the core of *i*-STS, built on the *N5* move operator. Short-term memory is used to prevent inversion of recently swapped pairs of adjacent operations on a critical path. Following Taillard (1989), the tabu tenure is periodically and randomly sampled from a fixed interval $[L, U]$. Search in *i*-STS proceeds in two phases. In the first phase, the basic tabu search algorithm is executed for a small, fixed number of iterations from each of a number of distinct random initial solutions. The best solution from each iteration-limited run is saved, and the aggregate forms the initial set E of elite solutions.

In the second phase of *i*-STS, the elite solutions in E are iteratively processed by both intensification and diversification mechanisms, each selected at any given iteration with respective probabilities p_i and p_d, where $p_i + p_d = 1$. To perform search intensification, a single elite solution $e \in E$ is selected at random and an iteration-limited tabu search is executed from e. Due to random tie-breaking during move selection, facilitated by the pervasiveness of plateaus of equally fit neighboring solutions in the JSP (Watson, 2003),

Algorithm 1: *i*-STS: Iterated Simple Tabu Search

```
*i*-STS():
1  initialize elite solution set $E$ via tabu search applied to random initial solutions
2  while termination criteria not met do
3      $p$ := draw random sample from the interval $[0, 1]$
4      if $p \leq p_i$ then
5          $e$ := draw solution at random from $E$
6          $e'$ := apply simple tabu search to $e$
7          if $C_{\max}(e') \leq C_{\max}(e)$ then
               replace $e$ in $E$ with $e'$
8      else
9          $e_1, e_2$ := draw two solutions at random from $E$, $e_1 \neq e_2$
10         $e'$ := apply path relinking between $e_1$ and $e_2$
11         $e''$ := apply simple tabu search to $e'$
12         if $C_{\max}(e'') \leq C_{\max}(e_1)$ then
               replace $e_1$ in $E$ with $e''$
13 return best($E$)
```

different search trajectories are generated. If a solution e' with a lower makespan than e is located, e' replaces e in E. To perform diversification, two elite solutions $e_1, e_2 \in E$ are selected at random. Path relinking is then performed to generate a solution e' that is approximately equi-distant from both e_1 and e_2. Iteration-limited tabu search is then executed from e', as is performed in the intensification process. If a solution e'' is identified with a lower makespan than e_1, then e'' replaces e_1 in E. The second phase of *i*-STS continues until a cumulative number of basic tabu search iterations M have been executed, with the best solution $e \in E$ returned upon completion.

With four exceptions, all parameters of *i*-STS are set identically to that reported in Watson et al. (2006). The exceptions are chosen based on the empirical studies of *i*-STS and *i*-TSAB (Watson et al., 2006) and are as follows:

- The probabilities of intensification and diversification are both set to 0.5 ($p_i = p_d = 0.5$).
- The number of iterations of tabu search allowed before the intensification either finds a new better solution or terminates is 7,000.
- Whenever the intensification phase does find an improved solution, this limit is reset to 20,000 iterations and the current tabu search continues executing.

- The elite pool size ($|E|$) is one of the independent variables in the experiments reported below.

2.3 Solution-Guided Search

Solution-guided search (SGS) is an algorithm that combines constructive tree search, randomized restart, and heuristic guidance from good solutions found earlier in the search (Beck, 2007). The basic approach is a CP tree search with a limit on the number of dead-ends ("fails") that are encountered before restarting. Each tree search is guided by using an existing sub-optimal solution as a value ordering heuristic. Once a variable to be assigned has been chosen (see below), the value chosen is the value of the corresponding variable in the guiding solution, provided that value is still in the domain of the chosen variable. Otherwise, any other value ordering heuristic may be used. As in i-STS, a small set of "elite" solutions is maintained, one of which is chosen with uniform probability to guide a given tree search. When a tree search exhausts its fail limit, it returns the best solution it has found (if any). That solution, if it exists, then replaces the guiding solution in the elite pool.

Beck (2007) showed that SGS has strong, but not state-of-the-art, performance on makespan-minimization JSPs. While finding significantly better solutions than chronological backtracking and randomized restart (using the same propagators, heuristics, and, in the latter case, fail limit sequences), SGS was not able to perform as well as i-STS.

2.3.1 Details

A simplified version of SGS is used in this paper. This version fixes a number of the parameters in the full algorithm. Readers interested in the full version are referred to Beck (2007).

Pseudocode for SGS is shown in Algorithm 2. The algorithm initializes a set E of elite solutions and then enters a while loop. In each iteration, a chronological backtracking search is guided with a randomly selected elite solution (line 6). If a solution s is found during the search, it replaces the starting elite solution r. Each individual search is limited by a fail bound: a maximum number of fails that can be incurred. The entire process ends when the problem is solved, proved insoluble within one of the tree searches, or when some overall bound on the computational resources (e.g., CPU time or number of fails) is reached.

More formally, a search tree is created by asserting a series of choice points of the form: $\langle V_i = x \rangle \vee \langle V_i \neq x \rangle$, where V_i is a variable and x is the value assigned to V_i. SGS can use

Algorithm 2: SGS: Solution-Guided Search

```
  SGS():
1 initialize elite solution set $E$
2 while not solved and termination criteria not met do
3     $r$ := randomly chosen element of $E$
4     set upper bound on cost function
5     set fail bound $b$
6     $s$ := search($r$, $b$)
7     if $s$ is better than $r$ then
8         replace $r$ with $s$ in $E$
9 return best($E$)
```

any variable ordering heuristic to choose the variable to assign. The choice point is formed using the value assigned in the guiding solution or, if the value in the guiding solution is inconsistent, a heuristically chosen value. Let a guiding solution r be a set of variable assignments $\{\langle V_1 = x_1\rangle, \langle V_2 = x_2\rangle, \dots, \langle V_m = x_m\rangle\}, m \leq n$, where n is the number of decision variables. Let $dom(V_i)$ be the set of possible values (i.e., the *domain*) of variable V_i. The variable ordering heuristic has complete freedom to choose a variable V_i to be assigned. If $x_i \in dom(V_i)$, where $\langle V_i = x_i\rangle \in r$, the choice point is made with $x = x_i$. Otherwise, if $x_i \notin dom(V_i)$, any value ordering heuristic can be used to choose $x \in dom(V_i)$.

At line 4 in the pseudocode, an upper bound is placed on the cost function for the subsequent search. We use the *local upper bound* approach here (Beck, 2007): the upper bound on the cost function is set to one less than cost of the guiding solution (i.e., cost(r)-1). Intuitively, the local upper bounding approach is a trade-off between exploiting constraint propagation (strongest if the upper bound were one less than the best solution we had found so far) and exploiting the heuristic guidance of high-quality but not necessarily best-so-far solutions.

Given a large enough fail limit (line 5), an individual search can exhaust the search space. Therefore, completeness depends on the policy for setting the fail limit. In our experiments, we use the Luby fail limit, an evolving sequence that has been shown to be the optimal sequence for satisfaction problems under the condition of no knowledge about the solution distribution (Luby et al., 1993). The sequence is as follows: 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ... Following Wu and van Beek (2007) and our own preliminary experimentation, we multiply the elements of the sequence by a fixed constant (in our case, 200). As the sequence increases without limit, a single search will eventually have a fail limit that is sufficient to search the entire search space and therefore the overall algorithm using the Luby fail limit is complete.

SGS is a general framework for constructive tree search. To apply SGS to the JSP, solutions are encoded using the well-known disjunctive graph representation. Texture-based heuristics (Beck and Fox, 2000) are used to identify a machine and time point with maximum contention among the operations and to then choose a pair of unordered operations. The heuristic is randomized by specifying that the ⟨machine, time point⟩ pair is chosen with uniform probability from the top 10% most critical pairs. The ordering found in the guiding solution is asserted. Note that because the decisions are binary, the pair in the solution must be locally consistent, otherwise the pair of operations would already be sequenced in the opposite order. The standard constraint propagation techniques for scheduling (i.e., edge finding, the precedence graph, and the timetable constraint) (Nuijten, 1994; Laborie, 2003; Le Pape, 1994), available via the ILOG Scheduler library (Scheduler, 2007), are also used.

2.4 Algorithm Selection and Switching

The algorithm selection problem, introduced by Rice (1976), is to identify the algorithm (or the particular parameterization of an algorithm) that has the best performance on a given problem instance. Algorithm performance on hard combinatorial problems, including the JSP, is known to be highly variable. Consequently, the ability to identify problem instance features and correlate these features to algorithm performance in an off-line learning phase can significantly accelerate solution times when the same features are used to select an algorithm on-line. Most of the work that has addressed this problem is what Carchrae and Beck (2005) have termed "high-knowledge": learning detailed models relating problem features and algorithm performance, e.g., see Minton (1996); Leyton-Brown et al. (2002); Boyan and Moore (2000); Lagoudakis and Littman (2000); Horvitz et al. (2001).

In contrast, Carchrae and Beck introduced a "low-knowledge", switching-base approach where there is no off-line learning but rather the control switches among the algorithms *during* problem solving and algorithms communicate their best-known solutions among each other. Over the course of the run, reinforcement learning is used to vary the time that the different algorithms are allocated so that the algorithms that have performed better receive an increased amount of the run-time. On a series of job-shop scheduling problems, Carchrae and Beck show that they are able to achieve on-line performance that is equivalent to the optimal off-line predictive approach.

3. Two Very Simple Hybrid Algorithms

Our work adopts the low-knowledge switching approach of Carchrae and Beck (2005). However, our motivation is not to compare low-knowledge and high-knowledge approaches nor to directly address the algorithm selection problem. Rather, our motivation is to investigate whether two strong but very different problem solving strategies can achieve state-of-the-art performance via very simple hybridizations. Our intuition, as noted above, is that tabu search and SGS have different behaviors in terms of intensification and diversification and that this difference may give rise to stronger combined performance.

We initially investigate two particular forms of hybridization:

- One-switch: Given an overall run-time limit of T seconds, the one-switch hybrid runs i-STS for $T/2$ seconds and then switches to SGS for the remaining $T/2$ seconds. As both algorithms use an elite pool, unlike Carchrae and Beck, the entire elite pool is transferred between the algorithms. That is, the elite pool at the end of the i-STS run is used as the initial elite pool for SGS (line 1 of the SGS pseudocode). The best solution found during the combined run is reported.

- Multi-switch: Using the same overall run-time limit T, a base iteration time $b \leq T$ is defined. Within each iteration, i-STS is run followed by SGS with the elite pool communicated as above. The next iteration then begins, again with i-STS but now initialized with the elite pool from the immediately preceding SGS run. The value of b, in addition to the relative proportion of b allocated to each algorithm, can either be static or can vary over the course of execution. We investigate a number of parameter settings below.

Obviously, one-switch is identical to multi-switch with $b = T$. We make the distinction primarily for clarity of presentation. The i-STS algorithm is run first in each interval because we observed that it performs better than SGS at quickly improving poor solutions such as the initial elite pool; this behavior is analyzed further in Section 8.

4. Plan of Experiments and Analysis

In the following sections, we conduct four experiments that examine the performance of various parameterizations of our hybrid approach and analyze why specific hybrid parameterizations work well and others do not. Based on the experimental results, we additionally compare our results to the state-of-the-art as published in the literature. Our experiments and hypotheses are as follows:

- Experiment 1: We conduct a fully crossed experiment comparing both one-switch and multi-switch to the underlying pure algorithms. Our hypothesis is that the hybrid algorithm will exhibit significantly better performance than the pure approaches. We also expect multi-switch to outperform one-switch.
- Experiment 2: We compare one-switch and multi-switch against variants of the hybrid algorithm that do not use SGS. Instead, they use chronological backtracking or randomized restart (Gomes et al., 2005). Our hypothesis is that the hybrid using SGS will outperform the other variations, demonstrating that SGS is critical for the performance observed in the previous experiments.
- Experiment 3: Following Carchrae and Beck (2005), we use reinforcement learning to vary the proportion of each time interval that is dedicated to each algorithm based on the performance of the algorithm during prior intervals. Our hypothesis is that by allocating more resources to the better performing algorithm, the reinforcement learning based approach will outperform static allocation over the interval.
- Experiment 4: We perform a controlled experiment to quantify the ability of the pure algorithms to improve upon an initial elite pool of a given quality. Here, our objective is to determine why specific parameterizations of our hybrid algorithm tend to work best.

Our experiments and analyses are based on multiple runs of each algorithm configuration on each of the Taillard benchmark instances we consider. Each problem instance is run 10 times independently for a given parameter configuration. *i*-STS is implemented in C++ while SGS uses ILOG Scheduler 6.5 (also in C++). All code was compiled using the GNU gcc compiler. Experiments were executed on a cluster with 2GHz Dual Core AMD Opteron 270 nodes, each with 2GB RAM running Red Hat Enterprise Linux 4.

5. Experiment 1: The Effects of Hybridization

Our first experiment is a fully crossed experiment that investigates the relative impact of all parameters and parameter combinations in our hybrid algorithm, with the goal of under-

Parameter	Value(s)	Meaning
T	3600	Total run time for one problem instance, in seconds.
$\|E\|$	$\{2, 4, 8\}$	The size of the elite pool for both *i*-STS and SGS.
a	{ists, sgs, hybrid}	The pure or hybrid algorithms.
b	$\{120, T\}$	The base iteration time in seconds. The first value corresponds to multi-switch, the second to one-switch.
g	{static, double, luby}	The iteration time growth sequence.

Table 1: The parameters for Experiment 1. See text for further details.

standing the relationship between these parameters and overall performance.

5.1 Parameters

Table 1 presents the parameters for the experiment. Parameters a and b interact to form the different pure and hybrid algorithms. When $a =$ hybrid, *i*-STS is executed for the first half of each iteration (i.e., $b/2$ seconds). On termination, *i*-STS returns its elite pool. SGS is then executed for the remainder of each iteration, starting with the elite pool from *i*-STS. At the end of each iteration, SGS returns its elite pool. In the next iteration (if any) *i*-STS starts from the elite pool returned by SGS in the previous iteration. If $b = T$, there is only one iteration. The $a =$ hybrid entries in Figure 1 present a schematic diagram of the hybrid algorithm when (1) $b = T$ and (2) $b = 120$.

When $a =$ ists or $a =$ sgs, the corresponding algorithm is executed for the full iteration time (i.e., for b seconds). At the end of each iteration, the algorithm terminates and returns its elite pool. The next iteration (if any) will then begin, starting with the elite pool from the previous iteration. For example, when $a =$ ists, the *i*-STS algorithm is repeatedly run for b seconds, restarting at the beginning of each iteration with the elite pool it found in the previous iteration. Figure 1 presents these variations. In all pure and hybrid schemes, the initial “elite” pool is constructed by executing *i*-STS for 15 seconds, achieved by severely limiting the number of iterations allocated to the individual tabu searches; consequently, the quality of the resulting solutions is generally poor.

The parameter g controls the *growth* in the iteration run-times over the course of a single run. Carchrae and Beck (2005) showed that doubling the iteration length after every iteration led to significantly better performance. Here, we experiment with three growth sequences: *static*, where all iterations are allocated b seconds; *double*, where each iteration run-time is double the length of the previous iteration, starting with b seconds for the first iteration;

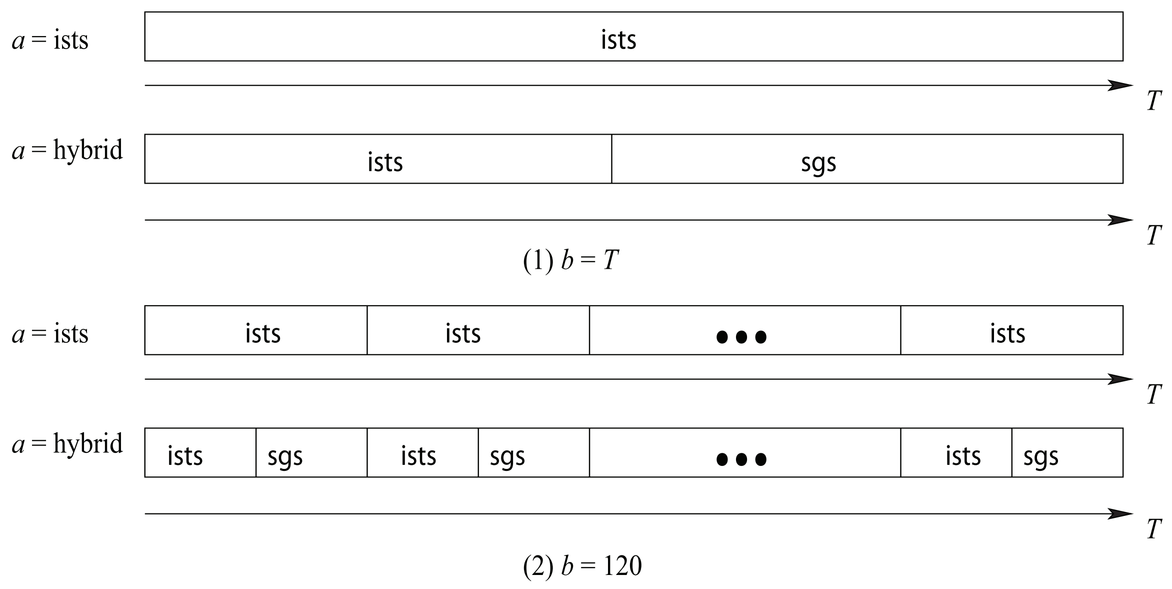

Figure 1: Schematic diagrams of the hybrid and pure algorithms for various values of the iteration run-time b.

and *luby*, where the iteration pattern follows the Luby sequence multiplied by b. This final strategy is motivated by the recognition that each iteration can be seen as a "restart" and therefore the theoretical results of Luby et al. (1993) apply.[1] It is worth noting, however, that our overall run-time limits prevent us from progressing very "deep" into the Luby sequence.

5.2 Results

We performed a two-way (factorial) ANOVA on the results.[2] The four independent variables are the elite pool size $|E|$, the algorithm a, the base iteration time b, and the iteration run-time growth sequence g. The sole dependent variable is the relative error of the best makespans obtained during a run. For a given problem instance and best solution makespan M, we define the relative error as $RE = (M - LB)/LB \times 100$, where LB is the largest known lower bound for the instance. The mean relative error (MRE) for a given parameterization is then computed simply as the mean RE taken over 400 data points: the 10 runs on each of the 40 problem instances. For our analysis, we take LB from Taillard (2008). The runs

[1]There are two independent uses of the Luby sequence that should not be confused. Within SGS, the Luby sequence (multiplied by 200) governs the change in the allocated fail limits of each tree search (line 5 in Algorithm 2). At the higher level, an independent Luby sequence (multiplied by b) determines the growth in the run-time of each iteration.

[2]All statistical analyses aside from the randomized pair-wise t-tests were performed using the R software package (R Development Core Team, 2006).

b	$\|E\|$	a	g	MRE	MBRE	MWRE
3600	8	hybrid	static	3.384	3.105	3.709
120	8	hybrid	double	3.395	3.108	3.724
120	8	hybrid	luby	3.434	3.135	3.772
120	8	hybrid	static	3.505	3.178	3.890
120	4	hybrid	double	3.519	3.176	3.931
3600	4	hybrid	static	3.521	3.163	3.928
120	4	hybrid	luby	3.551	3.169	4.009
120	4	hybrid	static	3.622	3.197	4.057
3600	2	hybrid	static	3.639	3.201	4.224

Table 2: The top nine parameter configurations based on mean relative error (MRE). Also listed are the corresponding mean best relative error (MBRE) and mean worst relative error (MWRE). The ranking of the configurations based on these statistics is almost identical to that using MRE.

for these experiments comprise over 1.5 years of CPU time.

An ANOVA of the results indicated significant effects at $p \leq 0.001$ for a, b, and their interaction. All other main and interaction effects were not statistically significant. We subsequently used the Tukey Honest Significance Difference (HSD) method with significance level $p \leq 0.005$ to compare the values of each variable. These tests indicated that $b = 120$ is significantly better than $b = T$ and that the hybrid algorithm achieves significantly lower MRE than i-STS which in turn achieves significantly lower MRE than SGS.

Table 2 provides summary statistics for the best (not necessarily in a statistically significant sense) parameter configurations. We also calculated the mean best relative error (MBRE) and the mean worst relative error (MWRE). For the former, for each problem instance we take the lowest RE found over the 10 runs and then, over the 40 problem instances, calculate the mean of those best relative errors. MWRE is analogously calculated. In Figure 2, the MRE over time is shown for the two pure algorithms, the one-switch hybrid algorithm, and the multi-switch hybrid algorithm.

5.3 Discussion

Our primary hypothesis was that a simple hybrid composed by switching between a pair of "pure" algorithms would generate results that are significantly better than the pure algorithms in isolation. This hypothesis is strongly supported by our results. The best pure algorithm ($b = 120$, $|E| = 8$, $a =$ ists, $g =$ static) achieves an MRE of 3.705, worse than

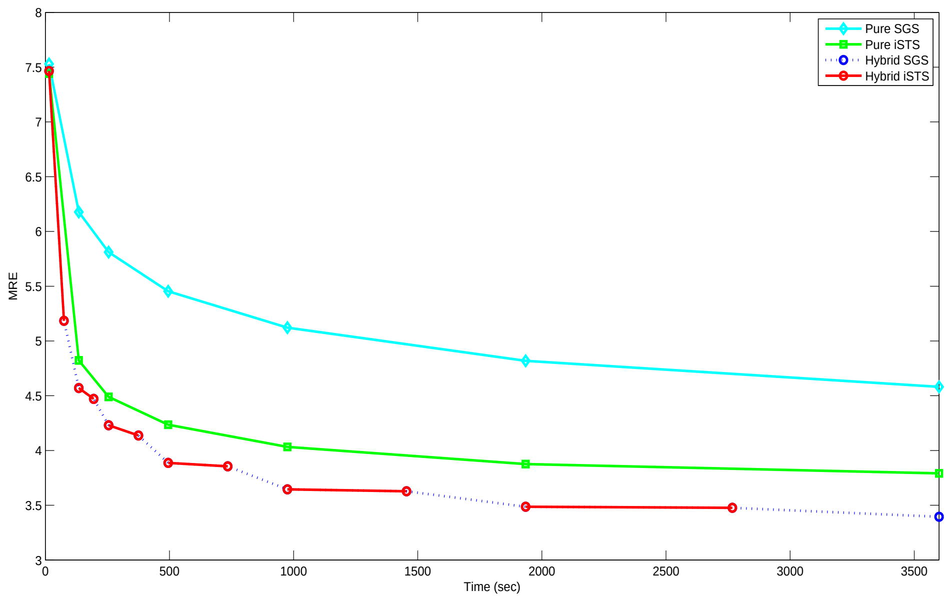

Figure 2: MRE performance over time for the two pure algorithms, the one-switch hybrid algorithm, and the multi-switch hybrid algorithm. For the hybrid algorithms, $|E| = 8$. For the multi-switch algorithm, $g =$ double.

the MRE of all configurations listed in Table 2. The results shown in Figure 2 graphically reinforce support for this hypothesis.

Our expectations that multi-switch would outperform single-switch is somewhat supported by the results. The ANOVA suggests that multi-switch (i.e., $b = 120$) is better than one-switch ($b = T$), however, as Table 2 indicates, three of the top nine configurations, included the overall best, are single-switch variations. A direct comparison of the top two configurations using randomized paired-t test (Cohen, 1995) showed no statistically significant differences at $p \leq 0.005$.

Our overall results contrast with Beck (2007), in which the elite pool size was a statistically significant factor, and to Carchrae and Beck (2005), where the g parameter governing the growth of the iteration time was statistically significant. However, these results are consistent with Watson and Beck (2008). The different experimental designs and algorithms used in these works lead to the following differences:

- Beck (2007) experiments only with SGS with an initial elite pool of random solutions. The i-STS algorithm is not used and there is no switching among the algorithms.

- Watson and Beck (2008) use *i*-STS and a variation of the one-switch hybrid algorithm. As such, this experiment is closest to the design and implementation of the current experiments and, in comparison to Beck (2007), the sole difference in the experimental designs is the mechanism used to initialize the elite solution set for SGS. It appears that while different parameterizations of SGS influence the degree to which the algorithm can improve upon random initial solutions (as shown in Beck (2007)), this sensitivity disappears once solution quality is "sufficiently" good, e.g., as is the case for *i*-STS solutions.

- Carchrae and Beck (2005) does not use *i*-STS nor SGS but rather experiment with switching between a tabu search algorithm and a pure constraint-based tree search algorithm. There is no elite pool and the only communication between algorithms is the best-so-far solution which tabu search uses as a starting solution and tree search uses only as an upper bound on solution cost. With richer communication (via the elite pool) and stronger pure algorithms, it appears that the significance of the iteration growth strategy disappears.

6. Experiment 2: The Impact of Solution Guidance

To evaluate the relative importance of SGS (as opposed to an alternative tree search strategy), we replaced SGS in our hybrid algorithm with both chronological backtracking and randomized restart. For completeness, we also experimented with running randomized restart and chronological backtracking stand-alone as we did with SGS and *i*-STS in Experiment 1. Guided by the results of Experiment 1, we performed a fully crossed experiment given the independent variables and associated values as shown in Table 3. We test a total of six core algorithms: SGS, randomized restart (rr), and chronological backtracking (chron) both alone and hybridized with *i*-STS as defined by the parameters a and c.

For both chronological backtracking and randomized restart, the upper bound on the makespan is one less than the cost of the best solution found by *i*-STS and the randomized texture-based heuristics and propagators described in Section 2.3 are employed. The value ordering in both cases is determined by the min-slack heuristic (Smith and Cheng, 1993). For chronological backtracking, there is no restarting of the search (i.e., the fail limit is infinite). For randomized restart, we used a Luby fail limit sequence with a multiplier of 200, as was used with SGS.

Parameter	Value(s)	Meaning
T	3600	Total run time for one instance, in seconds.
$\|E\|$	8	The size of the elite pool for both i-STS and SGS.
c	{sgs, rr, chron}	The constructive search algorithms.
a	{c, hybrid}	The pure-constructive or hybrid algorithms.
b	$\{120, T\}$	The base iteration time in seconds.
g	double	The iteration time growth sequence.

Table 3: The parameters for Experiment 2. See text for further details.

b	c	# SGS better	# equal	# SGS worse
3600	chron	38	2	0
	rr	36	3	1
120	chron	37	3	0
	rr	38	2	0

Table 4: A count of the number of problem instances for which the SGS version of the hybrid algorithm found better, equal, or worse mean relative errors over 10 runs than the hybrid algorithms using chronological backtracking and randomized restart.

A two-way ANOVA analysis of the results with $p \leq 0.001$ shows that a and b and their interaction are all statistically significant. Subsequent Tukey HSD tests ($p \leq 0.005$) show that (1) algorithms with $T = 120$ achieve significantly lower MRE than with $T = 3600$, (2) the hybrid algorithms are significantly better than the pure algorithms, and (3) c=sgs is significantly better than c=chron. However, no statistically significant difference is detected between c=sgs and c=rr or between c=rr and c=chron. In other words, our primary hypothesis – that SGS is important for the performance observed in Experiment 1 – is not supported by these statistical tests.

Looking at the results on individual problems, however, shows that with a single exception, the hybrid-sgs algorithm finds equal or better MRE values on each problem instance when compared with the MREs found by chronological backtracking or randomized restart algorithms. These counts are presented in Table 4. This result suggests that the lack of significance in the Tukey HSD tests arises from the inter-problem instance variance. Therefore, we performed four randomized paired-t tests (Cohen, 1995) to compare hybrid-sgs against the hybrid versions of chronological backtracking and randomized restart for each value of b. These tests show that the hybrid-sgs algorithm is significantly better than the corresponding hybrid algorithms for both b values at $p \leq 0.001$.

As a final indication of the contribution of SGS to hybrid algorithm performance, Table 5

b	ca	MRE	Mean # Improving Solutions
3600	sgs	3.384	6.02
	chron	3.844	0.04
	rr	3.828	0.02
120	sgs	3.395	21.30
	chron	3.792	0.04
	rr	3.805	0.01

Table 5: The mean relative error and the mean number of improving solutions found by hybrid algorithms using various tree search strategies in Experiment 2.

shows the mean relative error for each of the hybrid algorithms together with the number of times that each constructive algorithm found an improving solution. This latter value is the mean number of times in one run on one problem instance that the constructive search component (i.e., sgs, rr, or chron) was able to find a solution that improved on the current best solution in the elite pool. The table shows that SGS finds over two orders of magnitude more improving solutions than the other algorithms. The substantial differences between $b = 3600$ and $b = 120$ can be understood by recalling that in the $b = 3600$ condition, the tree search is run once, with a very good starting elite pool (i.e., the pool found by running i-STS for 1800 CPU seconds). In contrast, with $b = 120$, the tree search is run multiple times starting with a comparatively worse elite pool (i.e., initially, that found after running i-STS for only 60 seconds). Because the elite pool is of much higher quality in the former condition, there is significantly less opportunity for find improving solutions.

Our hypothesis was that SGS would outperform the hybrid algorithms using chronological backtracking and randomized restart. This hypothesis is strongly supported by our results, specifically those that account for the variance between problem instances. The randomized restart results are particularly interesting because the only difference relative to SGS is the guidance by an elite solution. We conclude, therefore, that elite solution guidance is *a* critical component of the hybrid's performance.

7. Experiment 3: Learning Run-Time Allocations

Thus far, we have restricted i-STS and SGS to share an equal proportion of the run-time b allocated to each iteration of the hybrid algorithm. However, it is possible or even likely that one algorithm may perform better in certain iteration regimes, e.g., early or late in the overall process. For example, Carchrae and Beck (2005) showed in the context of a switching-based hybrid that reinforcement learning (Sutton and Barto, 1998) is able to achieve stronger performance than the basic alternating hybrid approach by learning how to allocate variable proportions of later iterations to the different algorithms. The approach was to allocate iteration time based on an algorithm weight and to modify that weight based on the standard reinforcement learning equation: $w_{i+1}(a) = (1-\alpha) \times w_i(a) + \alpha \times p_i(a)$. Here $w_i(a)$ indicates the weight of algorithm a in iteration i and $p_i(a)$ is the normalized performance of algorithm a in iteration i. The weights of all algorithms in a given iteration are normalized to sum to 1 and the time in the iteration is directly proportional to the algorithm weights. As in Carchrae and Beck (2005), initially our algorithms are equally weighted and the performance in an iteration is calculated as the decrease in best-so-far makespan per second of run-time normalized by the total decrease in best-so-far makespan per second over the iteration.

In the context of low-knowledge switching-based hybrid model, Carchrae and Beck (2005) followed a random approach to select the order of the pure algorithms. In this section, because there are only two pure algorithms, we use two simple ordering approaches: run *sgs* after *ists* and run *sgs* before *ists*. The algorithm ordering parameter is h.

In order to evaluate the effect of reinforcement learning, we conducted an experiment that manipulates **the learning rate α, the iteration time growth pattern g, and the algorithm ordering approach h.** For the other parameters, we use the settings for best multi-switch algorithm from Experiment 1.[3] Table 6 presents the independent variables of our experiment. Note that **the assignment of** $\alpha = 0$ **and** $h = \{ists \rightarrow sgs\}$ corresponds to the non-learning algorithm as used in Experiment 1 where each pure algorithm is allocated half of each iteration.

A three-way ANOVA analysis of the results indicates that the learning rate α, the iteration time growth pattern g, nor the algorithm ordering approach h has any statistically significant effect. The best (albeit not statistically significant) MRE result, in fact, comes from the non-learning algorithm. Unexpectedly, this result fails to support our hypothesis that we would see improved performance when reinforcement learning was used to tailor the iteration time allocation. When we examine the evolution of the weight values during a run, we see a consistent reduction of i-STS weight (and therefore run-time) over time. This reduction is faster for larger α.

[3]The one-switch hybrid ($b = 3600$) can make no use of this learning as there is only one iteration.

Parameter	Value(s)	Meaning
T	3600	Total run time for one instance, in seconds.
$\|E\|$	8	The size of the elite pool for both i-STS and SGS.
a	hybrid	The algorithm.
b	120	The base iteration time in seconds, corresponding to multi-switch.
g	{double, luby}	The iteration time growth sequence.
α	$\{0, 0.1, ..., 0.9\}$	The learning rate.
h	$\{ists \rightarrow sgs,$ $sgs \rightarrow ists\}$	**The execution order of algorithms within one iteration.**

Table 6: The parameters for Experiment 3. See text for further details.

8. Experiment 4: Controlling Initial Solution Quality

To analyze why reinforcement learning of relative algorithm run-times in each iteration fails to improve hybrid algorithm performance, and why SGS is favored over i-STS in the reinforcement learning experiment, we next consider the ability of i-STS and SGS to improve on an elite pool of a *given* initial quality. In Experiments 1 and 2, for each run on each problem instance we recorded the contents of the elite pool whenever a new solution was added. For each problem instance, we extract all unique solutions from all runs, across all parameterizations, and sort the solutions from lowest to highest makespan. After ranking, the solutions are divided into the following bins based on the makespan percentile: **0515**, **2535**, **4555**, **6575**, and **8595**. The bin **2535**, for example, contains all solutions with a makespan between percentile 25 and 35; lower percentiles indicate high-quality solutions.

For reasons of simplicity and solution difficulty, we focus strictly on Taillard's problem instances `ta41-ta51`. For each instance, we independently and randomly sample 10 elite pools of cardinality 8 from each makespan percentile bin. We then run 10 independent replications of the pure i-STS and SGS algorithms on each starting elite pool. Each replication is executed for 100 seconds, as our experimental objective is to assess the ability of i-STS and SGS to improve upon an initial elite pool of fixed quality. In summary, we execute 5000 runs (10 instances *times* 5 bins *times* 10 elite pools *times* 10 runs) for each of the 2 algorithms. We measure the average MRE for each bin and algorithm, computed every 10 seconds.

The results are shown in Figure 3, which reports the average MRE versus time for both i-STS and SGS. First, we consider the results for the poorest initial quality, corresponding to bin **8595**. Here, while SGS outperforms i-STS in the first few seconds, i-STS dominates subsequently. Similar, but less dramatic, behavior is exhibited for bin **6575**. However, for higher-quality initial elite pools, we observe that SGS dominates i-STS, such that no performance cross-over point is observed. Overall, the results conclusively demonstrate that i-STS is able to rapidly improve the quality of an initially poor elite pool. However, its ability to continue to do so drops as elite pool quality improves, to the point where the benefits are minimal. In contrast, SGS encounters difficulty – we hypothesize, due to the underconstrained state – in improving poor-quality elite pools, but starts to dominate i-STS once a medium to high-quality elite pool is obtained. These patterns suggest that a fixed strategy emphasizing i-STS in early iterations and SGS in later iterations is the best switching design, and provides an explanation for why the one-switch algorithm performed well and for why reinforcement learning was both ultimately unable to learn a stronger strategy and de-emphasized i-STS over time.

9. Comparison with the State-of-the-Art

As is common in work of this kind, we now compare our algorithm's performance to that of the state-of-the-art for solving JSPs. We select two baselines for comparison: Nowicki and Smutnicki's i-TSAB tabu search algorithm (Nowicki and Smutnicki, 2005) and Zhang et al.'s hybrid tabu search / simulated annealing algorithm (Zhang et al., 2008). The i-TSAB algorithm represents the state-of-the-art from 2003 onwards, while Zhang et al.'s algorithm is a recently introduced competitor. A single "winner" is not easily determined, lacking carefully controlled experiments and availability of the source code of these two algorithms. However, it is clear from published performance analysis that these two algorithms are superior to all predecessors.

Table 7 compares the best configuration from Experiment 1 ($T = 3600$, $|E| = 8$, $a =$ hybrid, $g =$ static) against the two competing algorithms. Table 11 provides a detailed breakdown of results for each problem instance. We compute the MRE for the best-known solutions recorded in Taillard (2008) as of December, 2008 ("Best-Known"). Unfortunately, Nowicki and Smutnicki (2005) only report results for a single run of i-TSAB, complicating interpretation. Absent a rigorous alternative, we treat the corresponding results as representative of mean i-TSAB performance. The Zhang et al. statistics are taken over 10 independent runs of their algorithm on each problem instance. Without the actual sample populations, it is not possible to make statistical inferences regarding the relative performance of the Zhang et al. algorithm and our hybrid algorithm. Consequently, we proceed

Instance Subset	Best Known	*i*-TSAB	Zhang		Hybrid		
			Best	Mean	Best	Mean	Worst
ta11-20	2.29	2.81	2.37	2.92	**2.26**	**2.42**	2.69
ta21-30	5.38	**5.68**	**5.44**	5.97	5.50	5.70	5.89
ta31-40	0.46	0.78	0.55	0.93	**0.49**	**0.72**	0.98
ta41-50	4.02	**4.70**	**4.07**	4.84	4.17	**4.70**	5.28
Overall	3.04	3.49	**3.11**	3.67	**3.11**	**3.38**	3.71

Table 7: Performance statistics comparing *i*-TSAB, Zhang et al.'s hybrid tabu search / simulated annealing algorithm, and our hybrid on Taillard's benchmark instances. The "Best Known" column is based on the best-known upper bounds recorded by Taillard (2008). The bold entries indicate the best performance comparing "Best" to "Best" and "Mean" to "Mean".

with a qualitative analysis.

First, we compare the performance of our hybrid with that of *i*-TSAB. Overall, and on two of the four subsets (`ta11-ta20` and `ta31-ta40`), the hybrid outperforms *i*-TSAB in terms of MRE. On the other two subsets, the hybrid results are either the same (`ta41-ta50`) or only slightly worse (`ta21-ta30`). While the percentage advantage is small in absolute terms, we observe that due to the difficulty of these instances, apparently small differences have historically differentiated state-of-the-art algorithms from second-tier competitors. Although we cannot rigorously determine whether our hybrid performance dominates that of *i*-TSAB, it is clear that the performance is, *at a minimum*, indistinguishable. Again, we are treating the individual *i*-TSAB samples as representative of mean performance. If they are instead treated as a measure of the best performance, they are clearly worse that the hybrid best for all problem subsets.

Next, we compare the performance of our hybrid with that of Zhang et al.'s algorithm, hereafter referred to simply as Zhang's algorithm. In terms of MRE, the hybrid algorithm dominates the Zhang algorithm both overall and on each problem subset; overall, the advantage is 0.29%. In terms of mean best relative error, each algorithm dominates on two of the four problem subsets, with equal overall performance. Of particular interest is the excellent mean worst RE performance of our hybrid algorithm. On two of the subsets (`ta11-ta20` and `ta21-ta30`), the mean worst RE of the hybrid is better than the *mean* performance of the Zhang algorithm. Overall, the hybrid mean worst RE performance is only slightly worse than the Zhang MRE performance, with a difference of only 0.04%. Clearly, a significant advantage of our hybrid algorithm is the consistency of the state-of-the-art performance over

Instance Subset	# New Best	# Equal	# Worse
ta11-20	2	7	1
ta21-30	2	5	3
ta31-40	1	7	2
ta41-50	1	1	8
Overall	6	20	14

Table 8: The number of instances in each subset for which the best solution found by the best hybrid algorithm is better than, equal to, or worse than the previous best-known.

multiple runs, which is often elusive on very difficult benchmark problems.

A major issue in comparative assessment of state-of-the-art algorithms for the JSP involves quantification of computational effort. In addition to issues involving the use of disparate computing hardware, software engineering decisions and coding skill make such comparisons notoriously problematic. We do not address these issues here. Rather, we observe that from analyses of published performance reports (Nowicki and Smutnicki, 2005; Zhang et al., 2008), all three test algorithms were executed on modern computing hardware and the allocated run-times on the larger problem instances were all within a factor of three.

9.1 Best-Known Upper Bounds

Table 8 records the number of problem instances in each subset for which the best solution found by our best hybrid parameterization over its 10 runs is better than, equals, or is worse than the best-known solutions. As can be observed, this single parameterization of the hybrid algorithm is able to meet or improve upon the current best-known solutions in 26 of the 40 instances. This is an impressive result given that the best-known solutions are aggregated from a wide variety of algorithms rather than being found by a single algorithm.

Under all parameterizations our hybrid yielded ten new best-known solutions to Taillard's benchmark instances as shown in Table 9. Although our main research goal is not to enter "horse-race" competitions of the type that are particularly common in Operations Research (Hooker, 1996), the ability of an algorithm to establish new best-known solutions in a given domain is a common (albeit heuristic, because it fails to account for factors such as run-time, coding ability, machine, and related factors) benchmark for establishing the state-of-the-art in performance. At the very least, the ability of an algorithm to establish new best-known solutions with reasonable computing effort provides strong evidence of general effectiveness.

Instance	Prev. Best-Known	New Best-Known
ta11	1359	1357
ta19	1335	1332
ta21	1644	1642
ta24	1646	1644
ta26	1645	1643
ta40	1674	1673
ta41	2018	2010
ta42	1949	1947
ta49	1967	1966
ta50	1926	1924

Table 9: The makespan of new best-known solutions identified by the hybrid i-STS / SGS algorithms for Taillard's benchmark problems over a variety of algorithm parameterizations. They include the parameterizations where the run time, T, is set to 24 hours.

Hybrid Parameters	ta14	ta31	ta35	ta36	ta38	ta39
$\|E\| = 8$, $b = T$, $a =$ hybrid, $g =$ static	10	10	1	7	5	10
$\|E\| = 8$, $b = 120$, $a =$ hybrid, $g =$ double	10	10	2	10	4	10

Table 10: The number of runs (out of 10) for which the two strongest-performing parameterizations of our hybrid algorithm found an optimal solution and proved its optimality.

9.2 On Proving Optimality

Unlike previous state-of-the-art algorithms for JSP, our hybrid is a complete algorithm given a sufficiently large run-time limit T. It is therefore possible to both find an optimal solution and prove its optimality directly rather than based on previously known lower bounds.

Table 10 displays the number of runs (out of 10) for which the two strongest-performing parameterizations of our hybrid algorithm was able to prove optimality of the best solution located. That is, in each case, an individual tree search in SGS exhausted the search space without reaching its fail limit. We observe relatively consistent performance in proving optimality across different runs of the same parameterization and instance.

10. Discussion

This paper has demonstrated that a comparatively simple combination of a sophisticated tabu search algorithm and an advanced constraint programming constructive search is able to achieve state-of-the-art performance on a set of standard benchmarks for the job-shop scheduling makespan minimization problem. The reason for the comparatively simple hybridization is, of course, the fact that both pure algorithms use an elite pool of high-quality solutions to guide search. Therefore, it is easy to combine the algorithms by communicating the elite pool.

The use of an elite pool is not novel, indeed, the two state-of-the-art JSP algorithms from the literature discussed in Section 9 both use such a pool. However, there is little understanding of *why* elite pools enable such strong performance. As noted in Section 1, this work was motivated by informal ideas about differences in the search styles of the two foundational algorithms, specifically concerning intensification versus diversification. While our results are positive, it is important to note that this paper *does not test* these ideas. The ideas need to be examined through careful formalization and experimental design. If some sort of balance between intensification and diversification is posited, it is necessary to unambiguously define the two notions as well as the balance between them. Further, experiments that both measure and manipulate the intensification and diversification and demonstrate correlation and causation need to be performed. It is possible that there are other underlying explanations of our results, unrelated to these motivations. More rigorous testing of these ideas will be the focus on follow-on research.

11. Conclusions

Historically, the performance of constraint programming approaches – despite the availability of strong, domain-specific propagation and heuristic search techniques – has lagged that of local search algorithms on the classical job-shop scheduling problem. We introduced a family of simple hybrid algorithms that leverage the broad search capabilities of a high-performance tabu search algorithm for the JSP (i-STS) with the domain-specific inference capabilities of the state-of-the-art constraint programming algorithm for the JSP (SGS). The performance of the hybrid algorithm is at least competitive with the two state-of-the-art algorithms for the JSP: Nowicki and Smutnicki's i-TSAB tabu search algorithm and Zhang et al.'s hybrid tabu search / simulated annealing algorithm. While various factors outside our immediate control prevent us from making a more rigorous and precise statement regarding relative performance, we additionally observe that our hybrid algorithm was able to locate ten new best-known solutions to Taillard's notoriously difficult benchmark instances, providing additional evidence of the effectiveness of our approach. Further, our hybrid algorithm is able to *consistently* achieve excellent performance, e.g., the worst-case performance is roughly equivalent to the mean performance of the Zhang et al. algorithm.

While this paper focuses on the introduction and analysis of a hybrid algorithm in terms of performance, our original motivation was to better understand why constraint programming algorithms for the JSP – in particular, SGS – generally under-perform their local search counterparts. Although it is now clear that SGS has a niche relative to local search in state-of-the-art algorithms for the JSP, we have only begun preliminary investigations into understanding this niche and how SGS exploits it. For example, we have preliminary evidence that SGS acts primarily as an intensification mechanism for the elite solutions generated by *i*-STS, and is empirically more efficient than tabu search in that role. Overall, the present contribution establishes the hybrid *i*-STS / SGS algorithm as an interesting test subject; future research will analyze these and other questions raised by this performance analysis.

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Research Fund, Microway, Inc., and ILOG, S.A.. Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

Balas, E., A. Vazacopoulos. 1998. Guided local search with shifting bottleneck for job-shop scheduling. *Management Science* **44** 262–275.

Baptiste, P., C. Le Pape, W. Nuijten. 2001. *Constraint-based Scheduling*. Kluwer Academic Publishers.

Beck, J. C. 2007. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research* **29** 49–77.

Beck, J. C., M. S. Fox. 2000. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence* **117** 31–81.

Blażewicz, J., W. Domschke, E. Pesch. 1996. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research* **93** 1–33.

Boyan, J., A. Moore. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* **1** 77–122.

Carchrae, T., J. C. Beck. 2005. Applying machine learning to low knowledge control of optimization algorithms. *Computational Intelligence* **21** 372–387.

Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass.

Garey, M.R., D.S. Johnson, R. Sethi. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research* **1(2)** 117–129.

Glover, F., M. Laguna, R. Martí. 2003. Scatter search and path relinking: Advances and applications. F. Glover, G.A. Kochenberger, eds., *Handbook of Metaheuristics*. Kluwer Academic Publishers, 1–35.

Gomes, C. P., C. Fernández, B. Selman, C. Bessière. 2005. Statistical regimes across constrainedness regions. *Constraints* **10** 317–337.

Hooker, J. N. 1996. Testing heuristics: We have it all wrong. *Journal of Heuristics* **1** 33–42.

Horvitz, E., Y. Ruan, C. Gomes, H. Kautz, B. Selman, M. Chickering. 2001. A Bayesian approach to tackling hard computational problems. *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence (UAI-2001)*. 235–244.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* **143** 151–188.

Lagoudakis, M. G., M. L. Littman. 2000. Algorithm selection using reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 511–518. URL `citeseer.nj.nec.com/lagoudakis00algorithm.html`.

Le Pape, C. 1994. Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering* **3** 55–66.

Leyton-Brown, K., E. Nudelman, Y. Shoham. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP02)*. 556–572.

Luby, M., A. Sinclair, D. Zuckerman. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* **47** 173–180.

Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *Constraints* **1** 7–43.

Nowicki, E., C. Smutnicki. 1996. A fast taboo search algorithm for the job shop problem. *Management Science* **42** 797–813.

Nowicki, E., C. Smutnicki. 2005. An advanced tabu search algorithms for the job shop problem. *Journal of Scheduling* **8** 145–159.

Nuijten, W. P. M. 1994. Time and resource constrained scheduling: a constraint satisfaction approach. Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

R Development Core Team. 2006. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Rice, J.R. 1976. The algorithm selection problem. *Advances in Computers* **15** 65–118.

Scheduler. 2007. *ILOG Scheduler 6.5 User's Manual and Reference Manual*. ILOG, S.A.

Smith, S. F., C. C. Cheng. 1993. Slack-based heuristics for constraint satisfaction scheduling. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*. 139–144.

Sutton, R.S., A.G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Taillard, E. D. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* **64** 278–285.

Taillard, É.D. 1989. Parallel taboo search technique for the jobshop scheduling problem. Tech. Rep. ORWP 89/11, DMA, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Taillard, É.D. 2008. http://ina.eivd.ch/collaborateurs/etd/default.htm. December.

Watson, J.-P. 2003. Empirical modeling and analysis of local search algorithms for the job-shop scheduling problem. Ph.D. thesis, Department of Computer Science, Colorado State University.

Watson, J.-P. 2005. On metaheuristic "Failure Modes": A case study in tabu search for job-shop scheduling. *Proceedings of the Fifth Metaheuristics International Conference*.

Watson, J.-P., J.C. Beck. 2008. A hybrid constraint programming / local search approach to the job shop scheduling problem. L. Perron, M. Trick, eds., *Proceedings of the Fifth International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'08)*. 263–277.

Watson, J.-P., A.E. Howe, L.D. Whitley. 2006. Deconstructing Nowicki and Smutnicki's *i*-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers and Operations Research, Anniversary Focused Issue on Tabu Search* **33** 2623–2644.

Wu, H., P. van Beek. 2007. On universal restart strategies for backtracking search. *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming*. 681–695.

Zhang, C.Y., P. Li, Y. Rao, Z. Guan. 2008. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research* **35** 282–294.

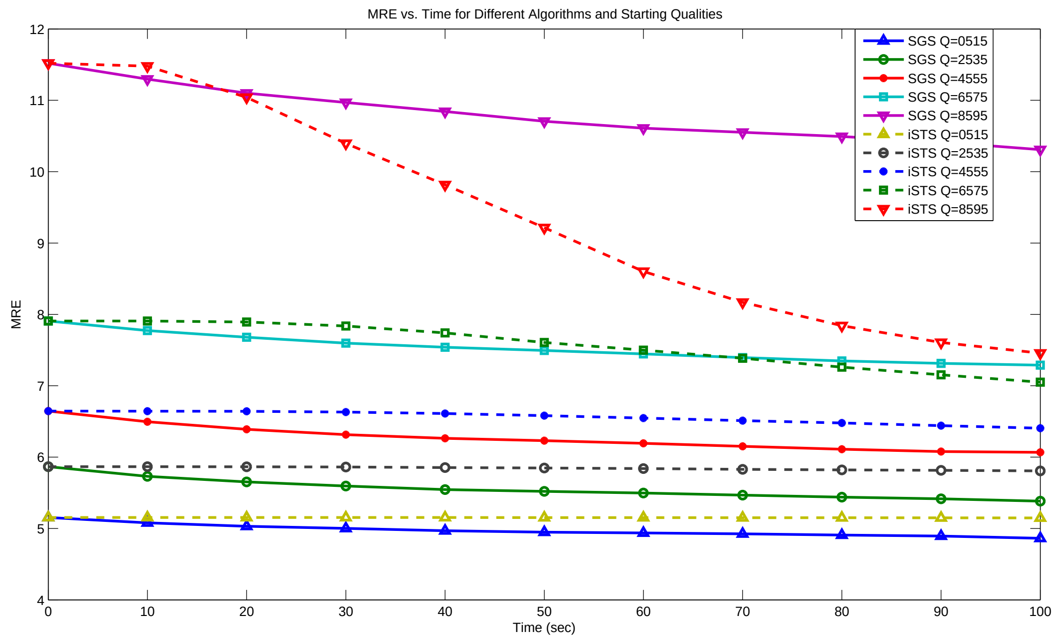

Figure 3: The relative performance of i-STS and SGS given starting elite sets of different quality.

			Best Hybrid		Second Best Hybrid	
Instance	LB	UB	Best	Mean	Best	Mean
ta11	1323	1359	**1357**	1362.1	**1357**	1362.7
ta12	1351	1367	1367	1369.5	1367	1369.4
ta13	1282	1342	1342	1343.3	1342	1345.5
ta14	1345	1345	1345*	1345.0	1345*	1345.0
ta15	1304	1339	1339	1339.0	1339	1339.0
ta16	1302	1360	1360	1360.0	1360	1360.5
ta17	1462	1462	1462	1463.8	1462	1463.8
ta18	1369	1396	1397	1401.0	1396	1401.1
ta19	1297	1335	**1332**	1333.6	**1332**	1333.4
ta20	1318	1348	1348	1353.0	1348	1353.8
ta21	1539	1644	**1642**	1644.4	**1642**	1645.5
ta22	1511	1600	1610	1613.1	1600	1610.6
ta23	1472	1557	1557	1559.2	1557	1558.9
ta24	1602	1646	**1645**	1647.6	**1645**	1646.6
ta25	1504	1595	1595	1601.4	1595	1601.1
ta26	1539	1645	1647	1648.8	1647	1650.2
ta27	1616	1680	1680	1684.1	1680	1685.5
ta28	1591	1603	1613	1615.7	1603	1614.3
ta29	1514	1625	1625	1626.3	1625	1626.4
ta30	1473	1584	1584	1588.7	1584	1589.7
ta31	1764	1764	1764*	1764.0	1764*	1764.0
ta32	1774	1795	1796	1809.1	1798	1806.5
ta33	1778	1791	1791	1796.3	1791	1798.5
ta34	1828	1829	1829	1831.0	1829	1831.0
ta35	2007	2007	2007*	2007.0	2007	2007.0
ta36	1819	1819	1819*	1820.7	1819*	1819.0
ta37	1771	1771	1777	1784.5	1774	1781.2
ta38	1673	1673	1673*	1675.1	1673*	1676.1
ta39	1795	1795	1795*	1795.0	1795*	1795.0
ta40	1631	1674	**1673**	1680.1	1674	1682.0
ta41	1859	2018	**2012**	2023.9	2018	2025.9
ta42	1867	1949	1956	1963.0	1957	1962.4
ta43	1809	1858	1863	1880.3	1863	1878.1
ta44	1927	1983	1991	1998.0	1992	1995.5
ta45	1997	2000	2000	2004.6	2000	2005.6
ta46	1940	2015	2016	2028.9	2023	2030.9
ta47	1789	1903	1906	1916.6	1910	1917.2
ta48	1912	1949	1951	1965.3	1956	1963.6
ta49	1915	1967	1969	1975.4	1971	1978.7
ta50	1807	1926	1932	1939.2	1934	1942.8

Table 11: The best and mean makespan found by the best and second best hybrid algorithms. Bold entries indicates new best makespans. '*' indicate that the optimal solution was found and proved without the use of the best-known lower bound.